Published online: 2020-08-10

APRICOT: Advanced Platform for Reproducible Infrastructures in the Cloud via Open Tools

Vicent Giménez-Alventosa[1] José Damián Segrelles[1] Germán Moltó[1] Mar Roca-Sogorb[2]

[1] Instituto de Instrumentación para Imagen Molecular (I3M), Centro mixto CSIC—Universitat Politècnica de València, Valencia, Spain
[2] Quantitative Imaging Biomarkers in Medicine (QUIBIM), Valencia, Spain

Address for correspondence Vicent Giménez-Alventosa, Instituto de Instrumentación para Imagen Molecular (I3M), Centro mixto CSIC—Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain (e-mail: vicent.gimenez@i3m.upv.es).

Methods Inf Med 2020;59:e33–e45.

Abstract

Background Scientific publications are meant to exchange knowledge among researchers but the inability to properly reproduce computational experiments limits the quality of scientific research. Furthermore, bibliography shows that irreproducible preclinical research exceeds 50%, which produces a huge waste of resources on nonprofitable research at Life Sciences field. As a consequence, scientific reproducibility is being fostered to promote Open Science through open databases and software tools that are typically deployed on existing computational resources. However, some computational experiments require complex virtual infrastructures, such as elastic clusters of PCs, that can be dynamically provided from multiple clouds. Obtaining these infrastructures requires not only an infrastructure provider, but also advanced knowledge in the cloud computing field.

Objectives The main aim of this paper is to improve reproducibility in life sciences to produce better and more cost-effective research. For that purpose, our intention is to simplify the infrastructure usage and deployment for researchers.

Methods This paper introduces Advanced Platform for Reproducible Infrastructures in the Cloud via Open Tools (APRICOT), an open source extension for Jupyter to deploy deterministic virtual infrastructures across multiclouds for reproducible scientific computational experiments. To exemplify its utilization and how APRICOT can improve the reproduction of experiments with complex computation requirements, two examples in the field of life sciences are provided. All requirements to reproduce both experiments are disclosed within APRICOT and, therefore, can be reproduced by the users.

Results To show the capabilities of APRICOT, we have processed a real magnetic resonance image to accurately characterize a prostate cancer using a Message Passing Interface cluster deployed automatically with APRICOT. In addition, the second example shows how APRICOT scales the deployed infrastructure, according to the workload, using a batch cluster. This example consists of a multiparametric study of a positron emission tomography image reconstruction.

Conclusion APRICOT's benefits are the integration of specific infrastructure deployment, the management and usage for Open Science, making experiments that involve specific computational infrastructures reproducible. All the experiment steps and details can be documented at the same Jupyter notebook which includes infrastructure

Keywords

- reproducible science
- life science
- cloud computing
- elasticity

received
February 13, 2020
accepted after revision
April 9, 2020

DOI https://doi.org/10.1055/s-0040-1712460.
ISSN 0026-1270.

specifications, data storage, experimentation execution, results gathering, and infrastructure termination. Thus, distributing the experimentation notebook and needed data should be enough to reproduce the experiment.

Introduction

Scientific publications are intended to share knowledge to be used by other researchers on their experiments. However, most publications do not provide the necessary information to verify and reproduce its results.[1] Moreover, Freedman et al[2] claims that "the cumulative prevalence of irreproducible preclinical research exceeds 50%," and Baker[3] shows that more than 90% of researchers consider that a reproducibility crisis exists in scientific publications. Nonreproducible science specially affects Life Sciences research, since these results should not be reused unless they can be trusted to avoid consequences on people's health. This results in a huge waste of resources on nonprofitable research.[2]

As a result, scientific reproducibility has lately become a topic of interest for researchers and institutions which defend an open and reproducible science model. This aims at solving two old problems in scientific research: nonreproducible investigation and fraud. With that purpose, the European Commission introduced Open Science to "promote a new approach to the scientific process based on cooperative work and new ways of diffusing knowledge by using digital technologies and new collaborative tools."[4] In fact, European institutions consider that Open Science is a necessary factor for future research programs. In 2015, the EU Commission set three main goals for future research in the EU: Open Science, Open Innovation, and Open to the world.[4,5] To promote these goals, the EU is promoting the European Open Science Cloud (EOSC), which "aims to create a trusted environment for hosting and processing research data to support EU science in its global leading role."[6] Open Science is not only a European objective, but also for institutions around the world. For example, in the United States, several institutions promote initiatives for Open Science, like the "Berkeley Initiative for Transparency in the Social Sciences,"[7] the Public Library of Science,[8] and the "Center for Open Science."[9]

We focus in this contribution on the reproducibility of computational experiments, where the two basic components required are *data storage* and *computing infrastructure*. The former, data storage, provides a place to store the experiment input, intermediate, and resulting data, analysis code and software, documentation, etc. The latter, computing infrastructure, is where calculations are performed according to the software and hardware requirements. This requires access to some local or external storage to stage in the required data for experimentation.

A general computational experiment workflow is represented in ►Fig. 1, where the *input data production* can be experimentally measured or simulated data, configuration values for simulations, input data files, etc. Then, the *data*

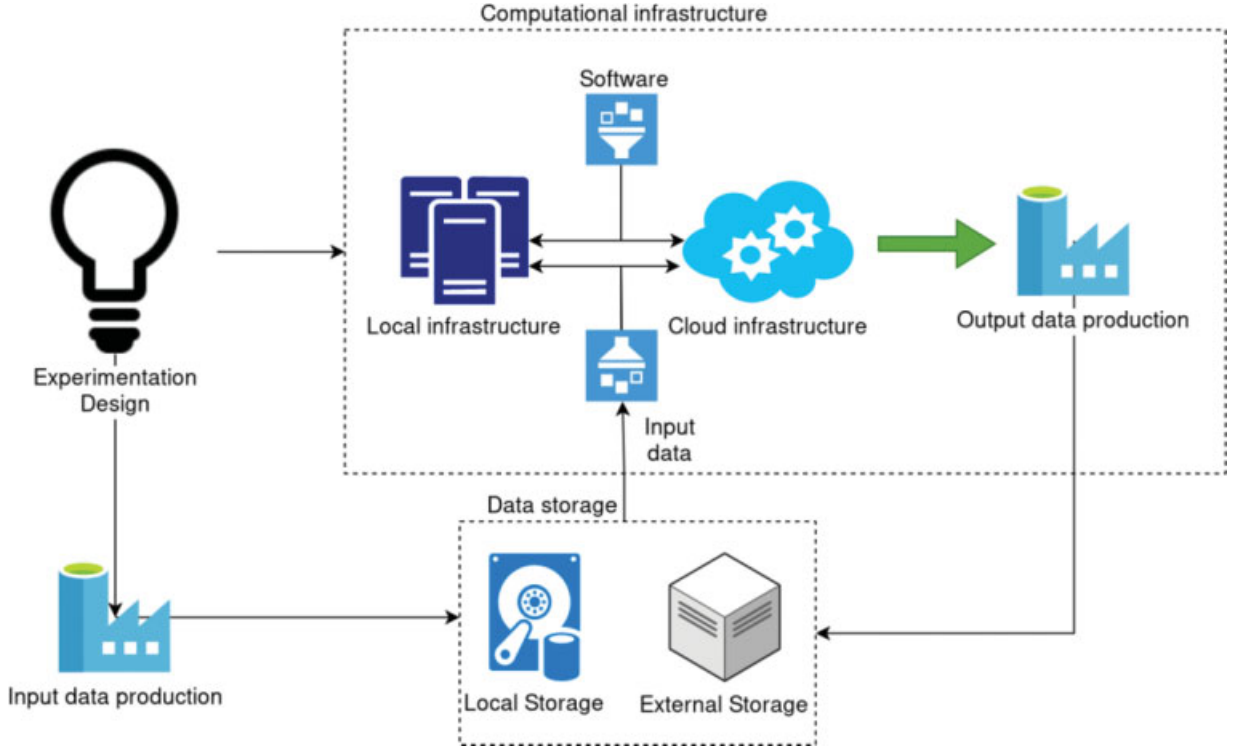

Fig. 1 Computational experimentation components.

storage can be provided not only by local storage but also by public cloud providers or open source storage platforms, such as Onedata.[10] To be able to reproduce an experiment, the required input data, software/code, and documentation must be accessible to other researchers via some external and persistent storage. Finally, it is at the *Computational infrastructure* where the input data are processed to produce the final result. This infrastructure largely depends on the computational experiment and, therefore, it can be composed of a single computer, a cluster of computing nodes, a set of clusters at different cloud providers, etc. In any case, the computational infrastructure requires a data storage to get input data and save both intermediate data, if needed, and the final results.

Also, the infrastructure requires some configured computational environment to perform the analysis, e.g., specific software. Some examples of research experiments in the field of Life Sciences following this pattern are described in the work by Chillarón et al,[11] which takes computed tomography (CT) measured data to construct a medical image with few projections. Other examples include the work by Reader et al,[12] which presents an algorithm to reconstruct positron emission tomography (PET) images with only one iteration over measured PET data and in the work by Giménez-Alventosa et al,[13] that uses radioactive sources specifications as input and perform brachytherapy simulations to discuss the error produced by approximations used at cancer treatment planning.

To achieve scientific reproducibility not only the data should adopt the FAIR principles (findable, accessible, interoperable, and reusable)[14] but also the software involved should follow similar principles. Software should be available in a public online repository that can be reached by means of a web search engine (Findable). It should be available under an open, nonrestrictive, software license to allow free adoption and modification to fit different purposes (accessible). It should provide application programming interfaces and command line interfaces using best practices widely established in the software development communities to ease the integration among multiple software services (interoperable). It should have a proper documentation that allows newcomers to start adopting the software and find areas of applications that perhaps even the original developer team had never thought of (reusable). The computational requirements to reproduce the experiment should also follow the same principles for scientists to self-deploy them to easily carry out the executions on a similar execution environment to guarantee a successful execution.

Even though some computational experiments may run on a regular computer, others may require a significant computing effort or specific customization, thus requiring a complex infrastructure. This is usually provisioned from private or public cloud providers. However, the deployment and configuration can involve a significant effort by the researcher and advanced technical knowledge. This increases the difficulty not only for researchers to reproduce published experiments, but also for reviewers, who cannot afford to spend the required time to reproduce the whole experiment.

Recently, many new platforms have been created to promote and facilitate the adoption of Open Science for researchers. These platforms offer different services such as cloud infrastructure to execute code in certain programming languages,[15,16] article journals and data repositories,[10,17,18] and shareable programming environments.[19,20] However, a lack of functionality stands out in these platforms related to the dynamic computational infrastructure deployment. Some experiments involve significant computational effort with specific distributed infrastructures or accelerated hardware devices, such as Message Passing Interface (MPI) clusters or GPGPUs. Current open science platforms cannot be used to reproduce these kind of experiments without a significant effort by the researcher and advanced technical knowledge such as infrastructure provisioning from multiclouds, understood as independent infrastructure deployments at different cloud providers, interconnection of nodes, and configuration of applications, job submission, etc.

To address this lack of functionality, this paper introduces APRICOT (Advanced Platform for Reproducible Infrastructures in the Cloud via Open Tools), an extension for Jupyter[19] notebooks to automatize the deployment, usage, and lifecycle management of virtualized computational infrastructures in multiclouds. We selected Jupyter because of its flexibility, ease of usage, capacity of module integration, and open source philosophy. APRICOT supports automatic deployment on multiclouds via a Jupyter plugin that uses both EC3 (elastic cloud compute cluster)[21] and IM (infrastructure manager)[22] to automatically provide and configure resources from multiple cloud back ends.

Objectives

The main aim of this paper is to improve reproducibility in the life sciences field to enable the production of better and more cost-effective research. For that purpose, our intention is to facilitate the infrastructure usage and deployment for researchers. Aimed at simplifying the researcher's technical effort, APRICOT offers a set of predefined cluster topologies such as "batch-cluster" or "MPI cluster." These are automatically configured with common utilities like a local resource management system (LRMS), shared home via network file system (NFS), a set of compilers, etc. and specific ones such as MPI libraries. Thus, researchers only need to specify their (temporary) credentials for the provision of resources from a cloud, the infrastructure topology, the number of nodes, and their characteristics (number of central processing units [CPUs], memory, etc.). Data management can be achieved via Secure SHell (SSH) and it also integrated with Onedata[10] to achieve on-demand caching and access to distributed datasets across providers.

APRICOT combines the computational narrative provided by Jupyter notebooks with the dynamic provision and integrated usage of virtual infrastructure from a cloud platform. This allows to create executable papers for reproducible science for research that involves specific computing requirements, which exceed those available in the researcher's computer.

Methods

This section has been divided in two major parts. First, we will discuss the previous-related work on reproducible

science to motivate the need of our application. Then, we will discuss APRICOT's architecture and how researchers can adopt this tool to simplify and make their computational experiments reproducible.

Related Work

The main tools for reproducible science can be divided in two big blocks: those focusing on data storage and those focusing on data processing. The latter is not restricted to provide a physical infrastructure where the application is run but also include tools that provide common and shareable computing environments.

On one hand, storage-oriented tools focus on the persistence of different kinds of elements for reproducible science where the researcher can upload the required data to reproduce a published experimentation. Also, we can find platforms to upload scientific research papers, open source code, etc. On the other hand, processing platforms provide physical infrastructure for data processing, platforms with implemented open source algorithms to be used by researchers, workflow pattern tools, and analysis workflows for specific problems, such as image recognition, etc. As we will see, most of them focus on creating easy-to-use shareable computational environments to promote reproducibility. Few of them provide simplified methods to deploy and configure virtualized computing infrastructure.

We can find in the literature code repositories for scientific reproducibility such as RunMyCode,[15] an online code, and data repository associated with scientific publications (articles and working papers). In this regard, the open science framework[23] helps sharing across institutions documents, materials, and data. Also, SEEK[24] is a web-based catalog for sharing scientific research datasets, models or simulations, processes, and research outcomes. These platforms do not provide the ability to remotely execute the code but rather to be downloaded for local execution. This prevents applications with specific computing requirements to be executed.

Scientific journals are starting to support reproducibility of results to some extent. This is the case of image processing on line (IPOL),[17] a research journal of image processing and image analysis. In IPOL, each article contains a description of the published algorithm and its source code, an online demonstration and a set of reproducible experiments. Text and source code are peer-reviewed, ensuring articles are truly reproducible. However, this is restricted to the topic of the journal: image processing.

There also exist open source tools such as the Galaxy project,[20] an open source workflow engine aimed at creating rapid and reproducible analyses that runs on an underlying infrastructure and can be used via a web browser. These workflows can be shared as documented experiments. This tool provides a common environment for researchers to work, facilitating software configuration and infrastructure usage. Another platform with similar features is the reusable and reproducible research data analysis platform called REANA.[25] REANA provides an environment to structure research input data and code using containerized environments and computational workflows allowing to instantiate and run the whole experiment on remote compute clouds, facilitating infrastructure usage. Experiments structured to be used with REANA can be easily reusable with the same input data, parameters, and code or changing them to obtain different results. These characteristics make REANA a suitable environment for Open Science in computational experiments. However, the infrastructure where Galaxy or REANA is installed must be deployed and configured in advance.

Jupyter[19] is an open source web application that allows users to create and share documents that contain live code, equations, visualizations, and narrative text. Documents, named "notebooks," can be easily shared and are used to create and share a complete experimentation workflow. JupyterHub provides multitenant access to Jupyter notebooks, introducing the ability to spawn notebooks, for example, on a Kubernetes cluster or on public cloud platforms. However, this automated provision of resources is limited to the execution of the Jupyter notebook. The provision of additional complex virtual infrastructures to support the execution of the computational experiment described in the notebook is responsibility of the user.

The aforementioned tools require external computing infrastructure to run the code. To alleviate this need, there exists completely managed reproducibility platforms, such as CodeOcean[16] a cloud-based computational platform where researchers can define a compute capsule that includes code and data together with the specification of a computational environment based on Docker. Users can execute all the published code without installing software on local computers. However, the free tier is limited and a subscription is required to unlock additional features. Managed platforms such as Stencila[26] allow researchers to create interactive, fully reproducible documents using familiar visual interfaces based on spreadsheets, thus restricting its applicability to cell-based calculations.

Existing reproducible science tools cover mainly the storage needs to provide easy and shareable environments for open science, as is the case of Jupyter notebooks or Galaxy instances. However, they do not provide a simplified method to deploy, configure, and use a personalized computing infrastructure from these environments. Even though some platforms like CodeOcean offer the possibility to execute code on a per-subscription basis, this approach is unfeasible for all kind of computational experiments and requires the users to lock-in to the platform. As mentioned before, some experiments involve significant computational effort and specific infrastructure characteristics, usually provided from private and public cloud providers, which require researchers to deploy a specific infrastructure and software configuration. Although there are tools to simplify the infrastructure deployment on multicloud platforms, they are not integrated in easy-to-use environments to encourage the use of these kind of infrastructures for reproducible science.

To address this issue we introduce APRICOT, an open source platform that extends Jupyter notebooks with the ability of self-provision of customized virtual computing infrastructure from multiclouds to execute the applications resulting from the simulation to be reproduced. We selected Jupyter notebooks as the base tool because of its flexibility,

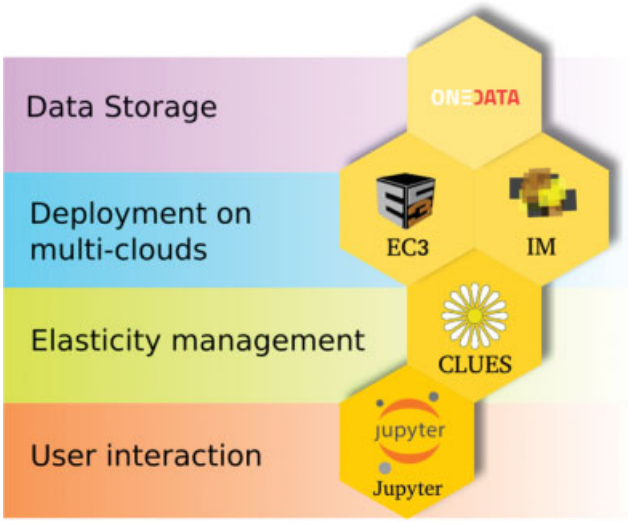

Fig. 2 Open source components used by APRICOT. APRICOT, Advanced Platform for Reproducible Infrastructures in the Cloud via Open Tools.

usage facility, capacity of module integration, and open source philosophy. This allows to combine in a single Jupyter notebook the description of the experiment with the ability of the user for the self-provision of customized complex computing infrastructure required to reproduce the results of the computational experiment.

APRICOT Architecture

APRICOT[a] is an open source extension to support customized virtual infrastructure deployment and usage from Jupyter notebooks. It allows multicloud infrastructure provisioning using a wizard-like graphical user interface (GUI) that guides the user step by step through the deployment process. For that purpose, APRICOT integrates several already existing open source components, which are summarized in ►Fig. 2. Each one performs a specific task to facilitate the infrastructure deployment on cloud providers. However, their standalone usage still requires advanced computational knowledge. Therefore, APRICOT has been designed to handle the communication between all these components and the user, automatizing the infrastructure deployment and fostering its usage among nonspecialized users. Next, we will briefly explain these components and their specific role:

- IM[22] is an open source tool that deploys complex and customized virtual infrastructures on Infrastructure as a Service (IaaS) cloud deployments, either public (such as Amazon Web Services [AWS], Google Cloud, or Microsoft Azure) or on-premises (such as OpenNebula and OpenStack). It automates the deployment, configuration, software installation, monitoring, and update of virtual infrastructures.
- CLUES (CLUster elasticity system)[27] is an elasticity system for high performance computing clusters and cloud infrastructures. Its main function is to deploy cluster nodes on the cloud when they are needed via dynamic provisioning and automated integration in the LRMS and, conversely, to terminate them when they have been idle longer than a certain time.
- EC3[21] is a tool to create elastic virtual clusters on top of IaaS cloud providers through the IM. To configure infrastructures, EC3 uses an infrastructure description language named RADL (Resource & Application Description Language) and, therefore, RADL files will be used by APRICOT. Multiple independent configuration files can be used in a single infrastructure deployment to create a complex configuration recipe.
- Onedata[10] is a global data access solution for science that provides access to distributed storage of scientific datasets, with automated caching and the ability to use multiple storage back ends. It can be used to store and share the data required to reproduce the experiments.

APRICOT's architecture aligns with the vision of the EOSC on the adoption of open source tools that can interoperate with the federated cloud infrastructures such as the one managed by EGI (European Grid Infrastructure). To this aim, both the IM and EC3 components were adopted, which are already integrated in the EOSC Marketplace[28] and are able to provide resources from the EGI Federated Cloud.[28]

As shown in ►Fig. 2, user interaction is managed via Jupyter notebooks. Then, the infrastructure deployment uses EC3[21] and IM[22] to support automated provision of computational resources from multiclouds. APRICOT provides its own Jupyter "magics" to manage the deployed infrastructure via the EC3 client, to perform data upload and download, to execute tasks, etc. These can be used in any kernel with "magic" commands support, thus being compatible with many programming languages in the Jupyter environment.

Concerning cluster elasticity, CLUES is automatically installed at the deployed virtual clusters to provide additional nodes when needed. Finally, Onedata are used as the default storage provider relying on an existing Onedata provider, such as those available to support the EGI Data Hub[b]. However, the user can adopt any external storage platform by installing the required client in the cluster front-end to upload and download data.

To develop a reproducible computational experiment using APRICOT, a researcher documents in a Jupyter notebook how the experimentation will be executed and how to obtain or generate the required data. Then, the researcher specifies the computing infrastructure requirements and topology, from a set of predefined topologies, which can be extended via additional RADL documents. Finally, all the commands to execute the experimentation should be documented and executed in the Jupyter notebook. To allow this requirement, APRICOT implements a set of IPython magic functions so that the executed instructions are actually performed in the deployed infrastructures, not in the execution environment provided by the Jupyter notebook.

[a] APRICOT - https://github.com/grycap/apricot].

[b] EGI Data Hub - https://datahub.egi.eu.

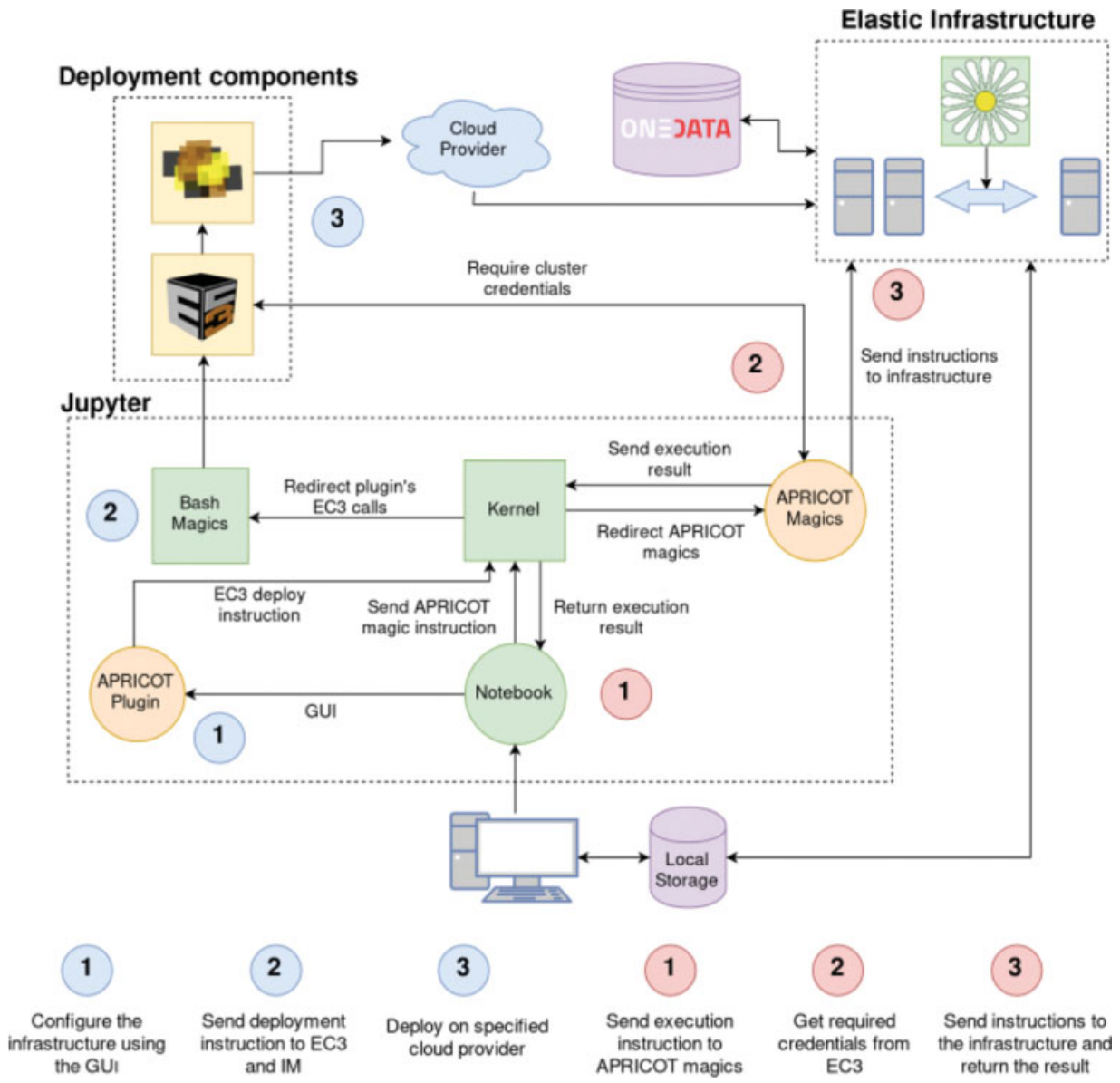

Fig. 3 APRICOT architecture. APRICOT, Advanced Platform for Reproducible Infrastructures in the Cloud via Open Tools.

Therefore, APRICOT is composed of an open source Jupyter notebook extension, used to deploy infrastructures, and a set of IPython magic functions for infrastructure use and management. The complete architecture schema is shown in ►Fig. 3.

First, APRICOT includes a GUI to guide the user on the infrastructure deployment process (►Fig. 4 shows the deployment steps and ►Fig. 5 shows screenshots of the GUI). The configuration includes the cluster topology, cloud provider, number of workers, etc. At the end of this step, the APRICOT plugin will instruct the EC3 client, using Bash Magic functions[c] to deploy the specified infrastructure. Then, EC3 will delegate on the IM provisioning of the virtual machines to the specified cloud provider to provide the infrastructure. Access credentials are automatically generated and stored by EC3 and, thus, APRICOT will contact EC3 to gather the appropriate credentials to perform remote command execution via SSH on the front-end node of the provisioned cluster.

To configure infrastructures, APRICOT uses a set of predefined RADL configuration files that describe the computing requirements, in terms of CPUs, random access memory (RAM), disk space, etc. and include Ansible[29] roles to perform the unattended installation of software. The used RADL files depend on the selected topology by the user. However, it is possible to use the EC3 client to reconfigure an existing cluster using additional RADL files. Alternatively, users can choose the "Advanced" topology option to manually select the infrastructure configuration using a set of RADL files.

APRICOT Magics implement IPython magic functions to manage and use deployed infrastructures. Most of these instructions will be executed in the infrastructure front-end through remote SSH. Since Magics can be executed in all kernels with this support, APRICOT instructions can be executed in any language interpreted for one of these kernels. However, as it happens with

[c] Bash Magics - https://ipython.readthedocs.io/en/stable/interactive/magics.html.

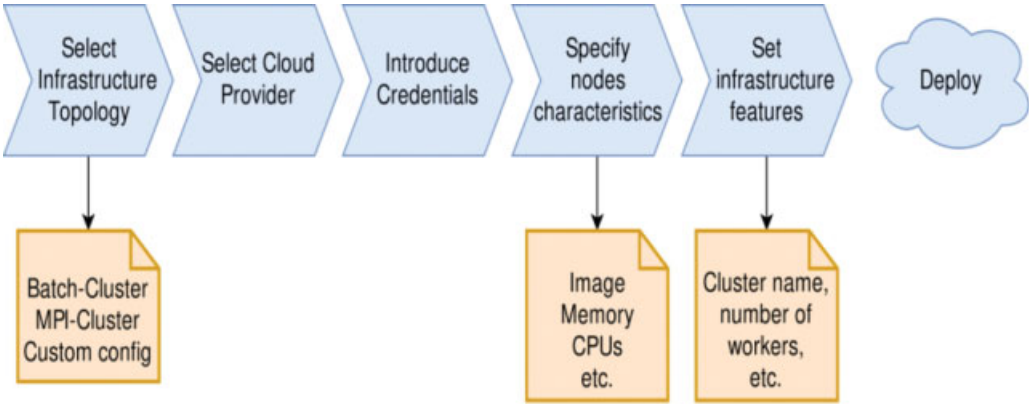

Fig. 4 Steps of the APRICOT deployment plugin. APRICOT, Advanced Platform for Reproducible Infrastructures in the Cloud via Open Tools.

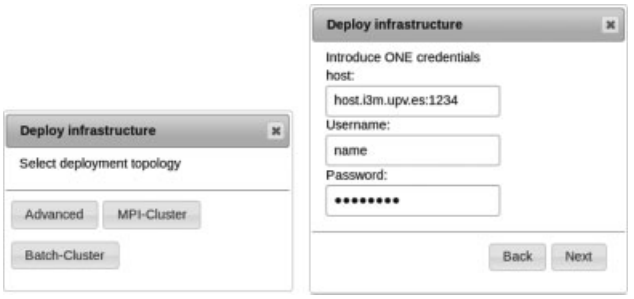

Fig. 5 Screenshots of the APRICOT deployment plugin. APRICOT, Advanced Platform for Reproducible Infrastructures in the Cloud via Open Tools.

all Jupyter nonbuilt-in Magics, APRICOT Magics must be loaded in each notebook before being used.

Notice that APRICOT just provides the infrastructure deployment, but the underlying computing infrastructure is not handled by APRICOT. So, if sensible data will be processed, the user must ensure that the infrastructure provider offers the required security measures. This is automatically enforced by major public cloud providers. Additionally, the user may decide to adopt additional risk mitigation strategies such as data encryption both at rest and in transit.

Finally, a Docker image is being maintained to create a ready to use containerized Jupyter server with APRICOT installed and configured, available in Docker Hub[d].

Results

To assess the effectiveness of the developed platform, this section introduces two reproducible experiments which are distributed alongside this document, publicly available as Jupyter notebooks in GitHub[e]. The examples are installed in the provided Docker image for the convenience of the reader. These demonstrates the complete execution of a real magnetic resonance imaging (MRI) processing and a multiparametric analysis for medical image reconstruction using APRICOT to deploy, use, and manage the required infrastructure. Therefore, this section represents a summary of the experimentation whereas further details can be obtained on the companion Jupyter notebook.

The aim of the first experimentation (MRI image processing) is to provide a real use case for our platform using a MPI cluster. On the other hand, the PET image reconstruction example uses simulated data and is intended to be used to evaluate APRICOT characteristics.

Infrastructure

To perform both experiments, we used an on-premises cloud managed by the OpenNebula[30] and the KVM hypervisor. The storage area network is a Dell Equallogic PS4210 with 16 TB available. The physical infrastructure constituted of two type of nodes. The first one had 240 GB of solid state disk, 64 GB of memory RAM, two Intel(R) Xeon(R) CPU E5–2683 v3 2.00GHz processors with 14 cores each one, two Ethernet network adapter of 1 Gbps, and another one of 10 Gbps. The second node type had 250 GB of solid state disk, 128 GB of memory RAM, two Intel(R) Xeon(R) CPU E5–2660 v4 2.00 GHz processors with 14 cores each one and three Ethernet network adapters, two of 1 Gbps, and a third of 10 Gbps.

On that on-premises cloud we deployed two different topologies. For the MRI example, we used an MPI cluster formed by one front-end and three working nodes. All the nodes have been configured with the same characteristics: one CPU and 4 GB of RAM with 20 GB disk space. On the other hand, for the image reconstruction experimentation, we used a virtual elastic batch cluster with one front-end and two initial working nodes. Both the front-end and the working nodes have the same characteristics: two CPUs, 2 GB of RAM, and 20 GB of disk storage.

The OS image used on each experiment is a plain Ubuntu with version 16.04 and 18.04 LTS, configured using the "MPI cluster" and "Batch Cluster" option from the APRICOT deploy plugin respectively. Also, we recommend to configure the cluster so that all the working nodes are pre-provisioned at deployment time to avoid the delay introduced when nodes are dynamically provided when jobs are submitted to the LRMS.

To further illustrate the multicloud features of the platform, we also reproduced the second experiment using the AWS public cloud provider with a cluster formed by one

[d] https://cloud.docker.com/u/grycap/repository/docker/grycap/apricot.
[e] https://github.com/grycap/apricot/tree/master/examples.

front-end and two working nodes. We used the "t2.small" (2 GiB and 1 vCPU) instance type to deploy the front-end node and two "t2.micro" (1 GiB and 1 vCPU) instances for working nodes. The cluster was deployed at the "us-east-1" region using an Ubuntu 16.04 AMI. Note that the selection of these instance types strictly responds to a cost saving strategy aiming to illustrate the ability of the platform to deploy on a public cloud. More powerful instance types would result in a significantly higher performance of the application.

All the required source codes to execute the experiment have been stored at the aforementioned GitHub repository alongside with the notebook to facilitate the reproducibility of this experiment. We stored input data in a public Amazon S3 bucket and not in OneData to facilitate access for the readers, since no access token is required to retrieve the data.

The following section introduces both experiments and discusses the obtained results.

MRI Image Processing

Prostate cancer (PCa) is the second most frequent malignancy (after lung cancer) in men worldwide, counting 1,276,106 new cases and causing 358,989 deaths (3.8% of all deaths caused by cancer in men) in 2018.[31] Early detection of PCa allows for appropriate management of the disease, and prognostic biomarkers can help clinicians make an appropriate therapeutic decision for each patient and avoid unnecessary treatment.[32]

Due to recent progress in imaging, and particularly in MRI, the so-called multiparametric MRI that combines T2-weighted imaging (T2W) with functional pulse sequences such as diffusion-weighted imaging or dynamic contrast-enhanced (DCE) imaging has shown excellent results in PCa detection and has become the standard of care to achieve accurate and reproducible diagnosis of PCa.[33,34]

Pharmacokinetic modeling of the DCE-MRI signal is used to derive estimates of factors related to blood volume and permeability that are hallmarks of the angiogenic phenotype associated with most cancers. The accuracy of DCE relies on the ability to model the pharmacokinetics of an injected tracer, or contrast agent, using the signal intensity changes on sequential magnetic resonance images.

The first pharmacokinetic model was proposed by Kety,[35] who described flow-limited tracer uptake in tissue. This was followed by several pharmacokinetic models proposed by Tofts et al,[36] Brix et al,[37] and Larsson et al.[38]

The majority of these models are based on the characterization of the contrast exchange rate between the plasma and the extracellular space through parameters such as K^{trans}, that represents the rate at which the contrast agent transfers from the blood to the interstitial space (indicating the tumor microcirculation), the reflux constant, K_{ep}, that reflects the rate at which the contrast agent transfers from the extravascular extracellular space back to the blood and the extravascular extracellular leakage volume fraction v_e, which predominantly reflects the percentage of contrast agent in the extravascular extracellular space.

The study of these parameters helps characterize PCa, so estimating them accurately and robustly is a fundamental step. These parameters are calculated using the Tofts model,[39] which is equivalent to the generalized kinetic model,[40]

$$\frac{dCt}{dt} = K^{trans}C_p - k_{ep}C_t \quad (1)$$

where interesting parameters are K^{trans}, which is the transfer coefficient between blood plasma and the compartment, and the extracellular extravascular fractional volume (v_e). Also, k_{ep} is defined as $k_{ep} = K^{trans}/v_e$, and C_t is the concentration of lesson tissue defined as $C_t = C_1\ v_e$, where C_1 is the leakage space concentration.

To solve Eq. (1), we use the same approach as given in Flouri et al.[41] Here, the model is restructured and expressed as a convolution as follows,

$$f(t) = K^{trans}\left(a(t) \otimes \frac{e^{-t/T}}{T}\right) \equiv K^{trans}\frac{1}{T}\int_0^t d\tilde{u}\, a(\tilde{u}) e^{-(t-\tilde{u})/T} \quad (2)$$

where $T = 1/k_{ep}$ and $a(t)$ function is an experimental measure, so is only available at discrete times. The previous model is evaluated for values of $T \neq 0$ by interpolating linearly between the measured values of $a(t)$. Instead, for $T = 0$ the result is $f(t) = a(t)$.

Our case study consisted of a prostate image with $256 \times 256 \times 56$ voxels and 30 time points for each one. We fitted this image using the Eq. (2) implemented at the provided code in the APRICOT repository, which uses the ROOT libraries from CERN.[42] This analysis was performed using a MPI cluster with three working nodes, reducing the total computation time almost a factor 3 compared with the same experimentation performed on a single node. As a sample of the result, ►Fig. 6 shows four images that represent the resulting v_e map of four planes.

PET Image Reconstruction

Medical scanners, like most physical detectors, measure raw data that must be post-processed to obtain an interpretable result. In particular, for medical scanners based in PET or CT, the final result is usually a patient image to be interpreted by the physician to develop a diagnostic.

Focusing on PET and CT cases, there exists a great variety of iterative and analytic image reconstruction algorithms,[11,12,43,44] most of them based on maximum likelihood method.[45] These reconstruction algorithms have a set of variable parameters such as number and size of voxels in the field of view, number of iterations, number of partitioned data chunks, weight parameters, filter iterations, and weight, etc. Obviously, the final reconstruction quality and speed will depend on how accurate are the selected parameters. Furthermore, the accuracy of selected parameters depends on the scanner system (geometry, energy resolution, scanned object, etc.). Indeed, the importance of reconstruction parameters on medical image has been studied for different kind of scanners in many publications.[46–49]

Achieving the best parameters for our specific system and algorithm is desirable not only for medical diagnostics but to perform accurate comparison of reconstruction methods and scanner capabilities. This comparison is crucial to select and create new scanner systems using simulated data to study

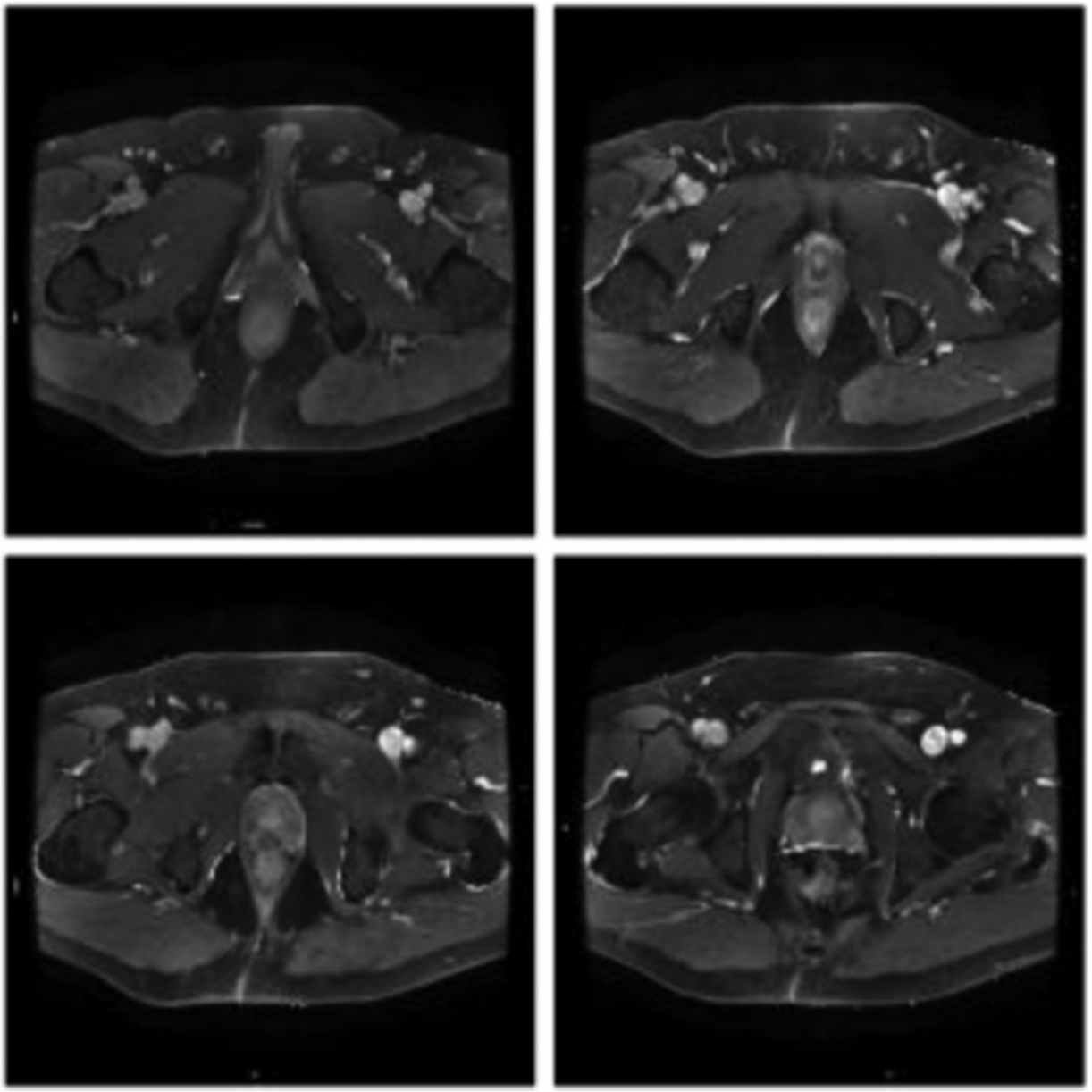

Fig. 6 MRI v_e maps for a real case of prostate image with 256 × 256 × 56 planes with 30 time points on each one. Images correspond to plane numbers 2 and 12 for top left and right images, respectively, and 22 and 32 for bottom left and right images, respectively. MRI, magnetic resonance imaging.

their theoretical performance. However, the number of possible parameters combinations grows as indicated in Eq. (3).

$$\prod_{i=1}^{n_{param}} N_i \quad (3)$$

where N_i is the number of possible values of parameter number i and n_{param} the number of variable parameters. So, even performing a multiparametric study with few parameters requires a significant computational effort. APRICOT has been used in this experimentation to deploy and manage the required infrastructure to perform a multiparametric study on a modified implementation of "OPL-EM" reconstruction algorithm for PET systems described in the work by Reader et al.[12]

This algorithm uses a single iteration over all measured data to reconstruct the image, thus being faster than other iterative algorithms. As the example is aimed to focus on APRICOT usage, a simplified experimentation will be reproduced in the companion material, for the sake of better understanding. The provided infrastructure will consist of a cluster of PCs configured with CPU-based working nodes. The use of accelerated devices such GPGPUs can be achieved by providing the corresponding instance types in a public cloud, provided that the application supports using such specialized hardware. Therefore, this is agnostic to the functionality provided by APRICOT.

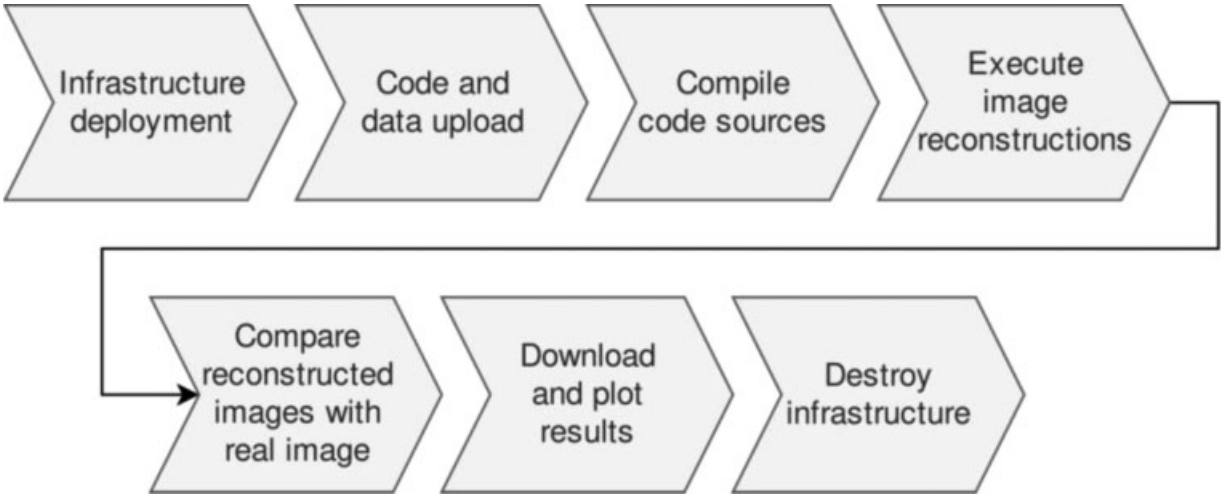

Fig. 7 Experimentation workflow.

The workflow of these experimentation is shown in ►**Fig. 7**.

The input data used for this experimentation has been obtained simulating a PET system formed by three rings of 20 detector modules each one. The simulations have been done using self-developed routines to perform PET system simulations with the Monte-Carlo code PENELOPE,[50] which accurately models photon, electron, and positron interactions in an arbitrary material for the energy range of interest in this work. The simulation results include a file with all photon detection grouped by coincidences, which means that both photons have been produced by the annihilation of the same positron. This file, which is provided to perform the experimentation, will be our reconstruction input.

Once the reconstructions have been done, we extracted their execution times and image quality measures using different parameter combinations. To measure the image quality, we used the following metrics, *root-mean-square error* (RMSE), *peak signal to noise ratio* (PSNR), *normalized root mean square distance* (NRMSD), and *normalized mean absolute distance* (NMAD), represented by Eqs. (4), (5), (6), and (7), respectively. Regarding the notation, $v(m)$ denotes the voxel number m of the considered image and the subindex *true* indicates that is the real image.

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{m=1}^{N}(v(m) - v_{true}(m))^2} \quad (4)$$

$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{\max(v_{true})^2}{\sqrt{\frac{1}{N}\sum_{m=1}^{N}(v(m)-v_{true}(m))^2}}\right) \quad (5)$$

$$\text{NRMSD} = \sqrt{\frac{\sum_{m=1}^{N}(v(m)-v_{true}(m))^2}{\sum_{m=1}^{N}(v_{true}-v_{true}(m))^2}} \quad (6)$$

$$\text{NMAD} = \frac{\sum_{m=1}^{N}|v(m)-v_{true(m)}|}{\sum_{m=1}^{N}|v_{true(m)}|} \quad (7)$$

The RMSE tends to zero when the reconstructed image and the ideal one coincide, because voxels at both images are equal. So, small values should be interpreted as better image quality. Next, PSNR tends to infinity when reconstructed and ideal image become equal, because RMSE tends to zero. The NRMSD metric tends to 1.0 when the differences between images are smooth and the average intensity equal. Large differences on few voxels produce a large value of NRMSD. Finally, NMAD tends to 1.0 if the reconstructed image has negligible intensity on its voxels respect to ideal image, and tends to zero if both images are equal.

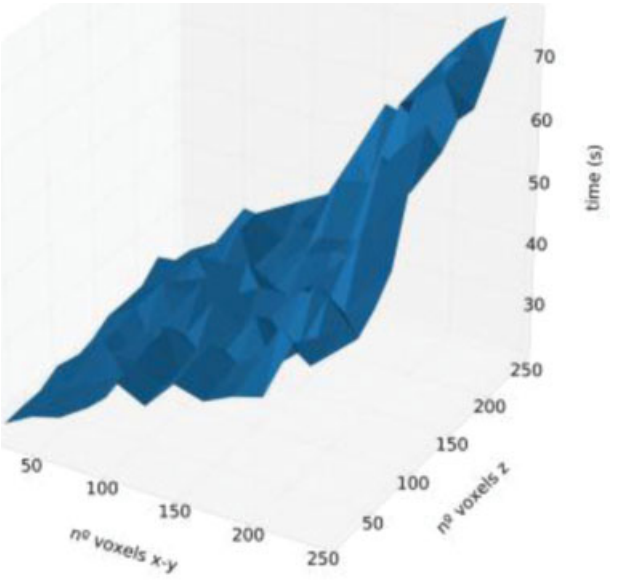

Fig. 8 Reconstruction time(s) using different number of voxels in each axis.

Some results of the run on our local infrastructure are shown in ►**Figs. 8** and **9** which represent, respectively, the time spent to reconstruct the images and the RMSE value of reconstructed images for different number of voxels in each axis using a data partitioning of five chunks.

In a real case study, these results should provide the best parameters to achieve the required agreement between reconstruction speed and image quality.

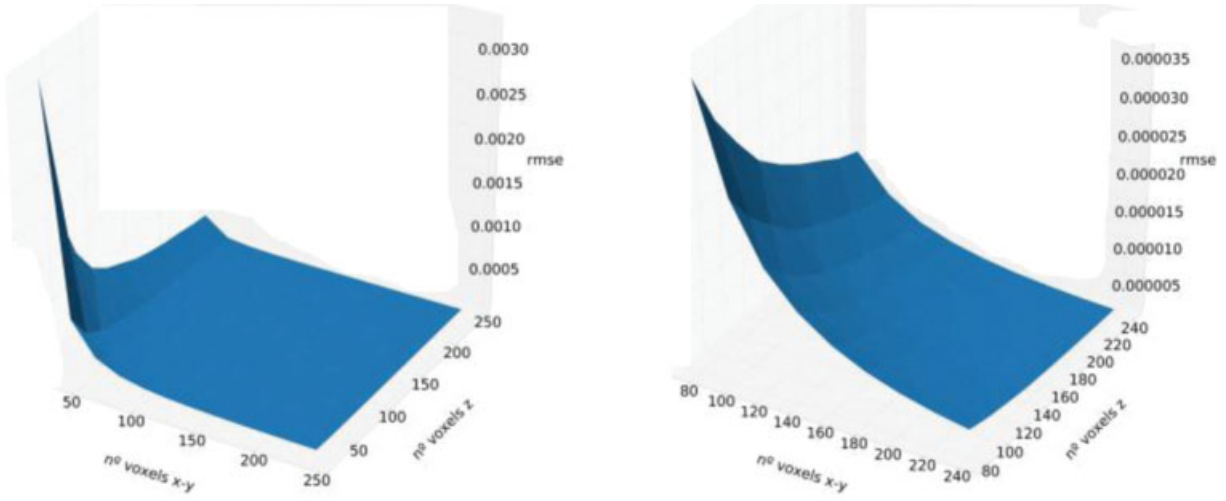

Fig. 9 Reconstruction error estimation using different number of voxels in each axis. The represented error has been obtained using the root mean square error (RMSE) method over all image voxels.

Discussion

This section discusses about the platform deployment and elasticity. For that purpose, the second experiment (image reconstruction) has been executed on two different platforms, according to the specifications in the subsection "Infrastructure" (under the "Results"), reproducing the very same results. The only difference lies on the location of the underlying resources and the time spent performing the reconstruction. The first one, deployed on our OpenNebula Cloud site, takes approximately 8 minutes to start and configure the front-end node and the same time for initial working nodes. On AWS, the deployment and configuration of the front-end node require approximately 10 minutes and the same for the working nodes. So, both clusters require, approximately, 16 to 20 minutes to be fully deployed and configured. Notice that working nodes are deployed concurrently. Therefore, deploying more working nodes will not cause a significant overhead on the configuration and deployment time. The main factor that affects the configuration time is the node CPU and network capabilities. Thus, using better instances should reduce the configuration time.

To show the elasticity capabilities provided by CLUES, we repeated the experiment using an infrastructure with the same specifications but configured with a minimum and maximum number of working nodes of 2 and 4, respectively. So, when the number of queued jobs exceed the number of available execution slots, CLUES deploys additional working nodes. To get statistics of the available and used slots we used CLUES reports, which monitors and extracts information about the use and state of our deployed infrastructure. ►**Fig. 10** shows a slots usage graph, where each node has a color identifier and the gray color represents idle slots. At the beginning of the graph (first gray zone) we can see when the first two nodes were configured and became ready to process jobs. Then, at the first colored zone, both nodes were filled with the received jobs and CLUES detected more queued jobs than available slots. This caused CLUES to power on the two extra nodes (second gray zone). Once all nodes had been powered on and configured, at the second colored zone, the number of available slots grew to four and the jobs execution was resumed. Finally, at the final gray zone, all the jobs had been processed and the working nodes became idle. Therefore, using CLUES, our deployed infrastructures can be automatically scaled within the specified configuration parameters.

As we have seen, APRICOT allows to easily deploy a scalable infrastructure to execute computationally intensive experiments. Furthermore, the same preconfigured infrastructure can be easily deployed to reproduce the whole experiment by other researchers or reviewers without

Fig. 10 Elasticity managed by CLUES. Cluster configured with two initial working nodes that can scale up to four nodes. The colored zones indicate the number of execution slots being used while the gray zone indicates an idle slot. CLUES, CLUster Elasticity System.

knowledge on infrastructure deployment and configuration. Without APRICOT, a researcher would need to deploy and configure the required infrastructure or have access to an existing one with compatible specifications.

At the moment, APRICOT offers a limited amount of preconfigured infrastructure types that may not fit all experiments need, but more configurations will be added in future versions. In addition, the user needs access credentials to a cloud provider supported by APRICOT.

Notice that the whole experiments have been documented in the corresponding Jupyter notebooks including the commands to execute, the required infrastructure, the data processing, the visualization, etc. The notebooks have been distributed in the aforementioned GitHub repository.

Conclusion

This paper has introduced APRICOT, an open source extension for Jupyter that provides users with the ability to deploy complex customized virtual infrastructures across multiple cloud providers to support the requirements of computational experiments. A set of functions have been created to simplify interaction with the virtual infrastructure for data staging as well as application execution. This facilitates the reproducibility of computational experiments on clouds.

The benefits of this extension are the integration of specific infrastructure deployment, the management, and usage for Open Science and making experiments that involve specific computational infrastructures reproducible. All the experiment steps and details can be documented at the same Jupyter notebook which includes infrastructure specifications, data storage, experimentation execution, results gathering, and infrastructure termination. Thus, distributing the experimentation notebook and the needed data should be enough to reproduce the experiment.

Future works include extending APRICOT to use additional cloud providers already compatible with the IM. Also, in addition to MPI and batch clusters we plan to add more preconfigured infrastructure topologies such as Kubernetes clusters. Regarding the infrastructure usage, we will provide more *magic commands* to simplify the execution of other kind of analysis. These are special Jupyter functions that can be executed regardless of the programming language used at the notebook. Integration of additional queue systems and external storage providers will also expand the adoption of the platform by covering different use cases.

Funding

This study was supported by the program "Ayudas para la contratación de personal investigador en formación de carácter predoctoral, programa VALi + d" under grant number ACIF/2018/148 from the Conselleria d'Educació of the Generalitat Valenciana and the "Fondo Social Europeo" (FSE). The authors would like to thank the Spanish "Ministerio de Economía, Industria y Competitividad" for the project "BigCLOE" with reference number TIN2016–79951-R and the European Commission, Horizon 2020 grant agreement No 826494 (PRIMAGE). The MRI prostate study case used in this article has been retrospectively collected from a project of prostate MRI biomarkers validation.

Conflict of Interest

None declared.

References

1 Donoho DL, Maleki A, Rahman IU, Shahram M, Stodden V. Reproducible research in computational harmonic analysis. Comput Sci Eng 2009;11(01):8–18
2 Freedman LP, Cockburn IM, Simcoe TS. The economics of reproducibility in preclinical research. PLoS Biol 2015;13(06):e1002165
3 Baker M. Is there a reproducibility crisis? A nature survey lifts the lid on how researchers view the 'crisis' rocking science and what they think will help. Nature 2016;533(7604):452–455
4 European Commission. Open Innovation Open Science Open to the World. European Commission; 2016
5 Goals of research and innovation policy. European Commission. Available at: https://ec.europa.eu/info/research-and-innovation/strategy/goals-research-and-innovation-policy. Accessed March 5, 2019
6 European Open Science Cloud (EOSC). European Commission. Available at: https://ec.europa.eu/research/openscience/index.cfm?pg=open-science-cloud. Accessed March 5, 2019
7 BITSS. Available at: https://www.bitss.org/. Accessed December 5, 2019
8 Public Library of Science. Available at: https://www.plos.org/. Accessed December 5, 2019
9 COS. Available at: https://cos.io/. Accessed December 5, 2019
10 Onedata. Available at: https://onedata.org/#/home. Accessed March 5, 2019
11 Chillarón M, Vidal V, Verdú G. CT image reconstruction with suite sparse QR factorization package. Radiat Phys Chem 2020;167:108289
12 Reader AJ, Ally S, Bakatselos F, et al. One-pass list-mode em algorithm for high-resolution 3-d pet image reconstruction into large arrays. IEEE Trans Nucl Sci 2002;49:693–699
13 Giménez-Alventosa V, Antunes PC, Vijande J, Ballester F, Pérez-Calatayud J, Andreo P. Collision-kerma conversion between dose-to-tissue and dose-to-water by photon energy-fluence corrections in low-energy brachytherapy. Phys Med Biol 2017;62(01):146–164
14 Wilkinson MD, Dumontier M, Aalbersberg IJ, et al. The FAIR guiding principles for scientific data management and stewardship. Sci Data 2016;3:160018
15 RunMyCode.org. Available at: http://www.runmycode.org/. Accessed March 5, 2019
16 CodeOcean. Available at: https://codeocean.com/. Accessed December 5, 2019
17 Ipol. Available at: https://www.ipol.im/. Accessed March 5, 2019
18 Nüst D, Konkol M, Schutzeichel M, et al. Opening the publication process with executable research compendia. Dlib Mag 2017;23(1/2):1082–9873
19 Kluyver T, Ragan-Kelley B, Pérez F, et al. Jupyter notebooks—a publishing format for reproducible computational workflows. Concurr Comput 2016
20 Galaxy. Available at: https://galaxyproject.org. Accessed: March 5, 2019
21 Calatrava A, Romero E, Caballer M, Moltó G, Alonso JM. Self-managed cost-efficient virtual elastic clusters on hybrid cloud infrastructures. Future Gener Comput Syst 2016;61:13–25
22 Caballer M, Blanquer I, Moltó G, de Alfonso C. Dynamic management of virtual infrastructures. J Grid Comput 2015;13:53–70
23 Open Science Framework. Available at: https://osf.io. Accessed: March 5, 2019
24 Wolstencroft K, Owen S, Krebs O, et al. SEEK: a systems biology data and model management platform. BMC Syst Biol 2015;9(01):33

25 REANA. Available at: http://www.reanahub.io/. Accessed November 4, 2019
26 Stencila. Available at: https://stenci.la. Accessed March 5, 2019
27 de Alfonso C, Caballer M, Calatrava A, Moltó G, Blanquer I. Multi-elastic datacenters: auto-scaled virtual clusters on energy-aware physical infrastructures. J Grid Comput 2019;17(01): 191–204
28 EOSC portal. Available at: https://marketplace.eosc-portal.eu/services/. Accessed October 30, 2019
29 Ansible. Available at: https://www.ansible.com. Accessed April 4, 2019
30 OpenNebula. Available at: https://opennebula.org/. Accessed: May 17, 2019
31 Rawla P. Epidemiology of prostate cancer. World J Oncol 2019;10 (02):63–89
32 Wu X, Reinikainen P, Kapanen M, Vierikko T, Ryymin P, Kellokumpu-Lehtinen PL. Dynamic contrast-enhanced imaging as a prognostic tool in early diagnosis of prostate cancer: correlation with PSA and clinical stage. Contrast Media Mol Imaging 2018; 2018:3181258
33 Bratan F, Niaf E, Melodelima C, et al. Influence of imaging and histological factors on prostate cancer detection and localisation on multiparametric MRI: a prospective study. Eur Radiol 2013;23 (07):2019–2029
34 Le JD, Tan N, Shkolyar E, et al. Multifocality and prostate cancer detection by multiparametric magnetic resonance imaging: correlation with whole-mount histopathology. Eur Urol 2015;67 (03):569–576
35 Kety SS. The theory and applications of the exchange of inert gas at the lungs and tissues. Pharmacol Rev 1951;3(01):1–41
36 Tofts PS, Wicks DA, Barker GJ. The MRI measurement of NMR and physiological parameters in tissue to study disease process. Prog Clin Biol Res 1991;363:313–325
37 Brix G, Semmler W, Port R, Schad LR, Layer G, Lorenz WJ. Pharmacokinetic parameters in CNS Gd-DTPA enhanced MR imaging. J Comput Assist Tomogr 1991;15(04):621–628
38 Larsson HBW, Stubgaard M, Frederiksen JL, Jensen M, Henriksen O, Paulson OB. Quantitation of blood-brain barrier defect by magnetic resonance imaging and gadolinium-DTPA in patients with multiple sclerosis and brain tumors. Magn Reson Med 1990; 16(01):117–131
39 Tofts PS, Kermode AG. Measurement of the blood-brain barrier permeability and leakage space using dynamic MR imaging. 1. Fundamental concepts. Magn Reson Med 1991;17(02):357–367
40 Donahue KM, Weisskoff RM, Burstein D. Water diffusion and exchange as they influence contrast enhancement. J Magn Reson Imaging 1997;7(01):102–110
41 Flouri D, Lesnic D, Sourbron SP. Fitting the two-compartment model in DCE-MRI by linear inversion. Magn Reson Med 2016;76 (03):998–1006
42 Rene Brun and Fons Rademakers. Root—an object oriented data analysis framework. Nucl Instrum Methods Phys Res A 1997;389 (1–2):81–86
43 Liu X, Comtat C, Michel C, Kinahan P, Defrise M, Townsend D. Comparison of 3-d reconstruction with 3D-OSEM and with fore+OSEM for pet. IEEE Trans Med Imaging 2001;20(08): 804–814
44 Singh S, Kalra MK, Hsieh J, et al. Abdominal CT: comparison of adaptive statistical iterative and filtered back projection reconstruction techniques. Radiology 2010;257(02):373–383
45 Shepp LA, Vardi Y. Maximum likelihood reconstruction for emission tomography. IEEE Trans Med Imaging 1982;1(02): 113–122
46 Goo JM, Tongdee T, Tongdee R, Yeo K, Hildebolt CF, Bae KT. Volumetric measurement of synthetic lung nodules with multidetector row CT: effect of various image reconstruction parameters and segmentation thresholds on measurement accuracy. Radiology 2005;235(03):850–856
47 Ravenel JG, Leue WM, Nietert PJ, Miller JV, Taylor KK, Silvestri GA. Pulmonary nodule volume: effects of reconstruction parameters on automated measurements—a phantom study. Radiology 2008; 247(02):400–408
48 Hu Y-H, Zhao B, Zhao W. Image artifacts in digital breast tomosynthesis: investigation of the effects of system geometry and reconstruction parameters using a linear system approach. Med Phys 2008;35(12):5242–5252
49 Lyra M, Ploussi A. Filtering in SPECT image reconstruction. Int J Biomed Imaging 2011;2011:693795
50 Salvat F, Fernández-Varea JM, Sempau J. Penelope. A Code System for Monte Carlo Simulation of Electron and Photon Transport. Issy-Les-Moulineaux: OECD Nuclear Energy Agency; 2014